Mixing Compression and CA Encryption

Bruno Martin

I3S, Université de Nice Sophia-Antipolis, CNRS,
2000 route des Lucioles, B.P. 121,
F-06903 Sophia Antipolis.
Bruno.Martin@unice.fr

We consider the enciphering of a data stream while it is being compressed by a dictionary algorithm. This scheme has to be compared to the classical encryption after compression methods used in classical security protocols. Actually, most cryptanalysis techniques exploit patterns found in the plaintext to crack the cipher and compression techniques reduce these attacks. Our scheme is based on a LZ compression algorithm in which a Vernam cipher has been included. We first present an experiment with a pseudo-random sequence generated by a simple linear feedback shift register before considering pseudo-random sequences generated by cellular automata. We make some remarks on the security by trying to measure the randomness of our method with some statistical tests. Such a scheme could be employed to increase the speed of some security protocols and to decrease the computing power for mobile devices.

Keywords: pseudo-random sequences, compression, one-time pad, cellular automata.

1 Introduction

Information security is currently one of the main challenges in the area of computer networks. In the emergent communication paradigm where wireless and wired networks are interoperating, security issues become crucial. Traditional technologies are every day more inadequate and existing standards should be improved for use in resource restricted environments. We aim to develop a secure algorithm for confidentiality, but cheaper in terms of bandwidth, speed and computing power.

In many security protocols, a compression algorithm is run prior encrypting the data to increase the security and the bandwidth. These algorithms are run on the original stream. They all stem from research by Lempel and Ziv who have designed two compression algorithms: LZ77 and LZ78. After compression, if the tradeoff between security and speed is taken into account, the compressed data is enciphered with the use of a stream cipher like for instance RC4 (let us recall that RC4 is 15 times quicker than a 3DES and is used in protocols like WEP and SSL [Res01]). We assume the reader familiar with classical compression algorithms and with secret key cryptography for which good introductions are [Mar04, Sal98, Sta06].

In the present paper, we propose to scramble (encipher) a data stream while it is being compressed by a lossless dictionary algorithm. The basic idea which motivates this proposition is that a compressed stream is almost pseudo-random and a good candidate to be combined with a simple Vernam cipher. This comes from the notion of incompressibility introduced by Kolmogorov complexity. Kolmogorov [Kol65] has proposed a complexity which speaks about objects rather than the usual classes of languages addressed by classical complexity. Informally, Kolmogorov complexity corresponds to the size of the smallest program p (denoted by $\sharp p$) which can print out on its standard output the object x. If $\sharp p < \sharp x$, we say that x is *compressible*, otherwise *incompressible*. It provides a modern notion of randomness dealing with the quantity of information in individual objects. It says that object x is random if it cannot be represented by a shorter program p whose output is x or, in other words, if x is incompressible [Sal98, LV93]. From this point of view, the output of a compression algorithm is close to a pseudo-random sequence, although highly reversible.

We then consider two mechanisms for generating the pseudo-random sequences which are used as keys for the Vernam cipher. We have made a first experiment with a linear feedback shift register. This test has been implemented and we have measured the quality of the output. We have obtained output sequences

which are apparently random although further testing with a χ^2 test was a little bit too weak. Thus, in order to improve the pseudo-randomness quality and to try to obtain a high rate generator, we consider in a second time pseudo-random sequences generated by cellular automata.

The paper is organized as follows: Section 2 recalls the definition of pseudo-random sequences. Section 3 presents our method with the test implementation. We use a compression algorithm from the Lempel-Ziv family and a pseudo-random sequence generated by a simple linear feedback shift register. Some statistical tests have been made and are presented in Section 3.4. In Section 4, we propose to increase the quality of the pseudo-random sequence. Instead of using a simple linear feedback shift register, we propose to replace it by a pseudo-random generator based on a cellular automaton, as proposed by Wolfram in [Wol85].

2 Generating (pseudo-)randomness

This section recalls the classical definitions of pseudo-randomness. We first give an intuitive statement which gives the difference between real randomness and pseudo-randomness. We then introduce more formal definition of pseudo-randomness coming from complexity theory.

In the book *A New Kind of Science*, Wolfram describes three mechanisms responsible for random behavior in systems:

- *Randomness coming from the environment* (e.g., brownian motion, also hardware random number generators);
- *Randomness coming from the initial conditions*. This aspect is studied by chaos theory, and is observed in systems whose behavior is very sensitive to small variations in initial conditions (such as pachinko machines, lottery, dices ...);
- *Randomness intrinsically generated by the system*. Also called pseudo-randomness; it is the kind of randomness used in pseudo-random sequences generators. Many algorithms (based on arithmetics or cellular automata) generate pseudo-random sequences. The behavior of the system is fully determined by knowing the seed and the algorithm used. They are quicker methods than getting "true" randomness from the environment, inaccessible for computers.

The applications of randomness have led to many different methods for generating random data. These methods may vary as to how unpredictable or statistically random they are, and how quickly they can generate random sequences. Before the advent of computational pseudo-random sequences, generating large amount of sufficiently random numbers (important in statistics and physical experimentation) required a lot of work. Results would sometimes be collected and distributed as random number tables or even CD iso-images.

More formally, a pseudo-random sequence can be defined as:

Definition 1 *A sequence is* pseudo-random *if it cannot be distinguished from a truly random sequence by any efficient (polynomial time) procedure or circuit.*

Theorem 1 ([BM84]) *A sequence is* pseudo-random *if and only if it is next-bit unpredictable.*

Theorem 1 claims that for pseudo-random sequences, even if we know all the previous terms of a sequence, we don't have any information on the next bit. Theorem 1 was proved equivalent to:

Theorem 2 ([Yao82]) *A pseudo-random sequence generator G passes Yao's test if, for any family of circuits F with a polynomial number of gates for computing a statistical test, G passes F.*

3 A first experiment

In this section we discuss our method which combines a Lempel Ziv compression algorithm together with a Vernam cipher which uses as key a pseudo-random sequence generated by a linear feedback shift register. We start with the description of two relevant ideas.

3.1 Related work

Actually, there are two methods sharing the same idea but in a slightly different way. The first one is called *concryption* and has been patented by Security Dynamics under US Patent #5479512. It is a method for the integrated compression and encryption (concryption) of clear data. For concryption, the clear data and an encryption key are obtained, at least one compression step is carried out and at least one encryption step is performed utilizing the encryption key. The encryption step is preferably performed on the final or intermediate results of a compression step, with compression being a multistep operation.

The second method is called *comcryption* and is due to Crandall [Cra98] when he was Apple's Chief Cryptographer. Roughly speaking, his idea is to index a great number of entropy compression algorithms by a secret key. He then gets a holistic (one-pass) compress/encrypt algorithm. This method is currently used for enciphering the passwords in the `keychain` application starting with Mac OS 9 and still used in Mac OS X from Apple Inc. It is recorded under US Patent #6154542, "Method and apparatus for simultaneously encrypting and compressing data".

3.2 The proposed scheme

We use the mode of operation of LZ 78 which uses a growing dictionary [Sal98]. It starts with $2^9 = 512$ entries (with the first 256 entries already filled up, eventually after an initial permutation). While this dictionary is in use, 9 bit pointers are written onto the output stream and directly encrypted by a Vernam cipher. When the original dictionary is filled up, its size is doubled to 1024 entries and 10 bit pointers are then used (and encrypted as well) until the pointer size reaches a maximum value set by the user. When the large dictionary is filled up, the program continues without changes to the dictionary but with monitoring the compression ratio. If this ratio falls down a predefined threshold, the dictionary is deleted and a new 512 entries dictionary is started.

The difference with concryption is that we use a single pass compression algorithm while concryption requires the compression to be a multistep operation. Our method is not based on an entropy compression algorithm used in the comcryption method, although an entropy compression algorithm could be added to shorten the mostly used pointers which are returned by the algorithm.

Compared with the classical compression prior encryption used by the classical security protocols, ours does not need to wait until the compression process stops (which is a costly operation) to start the encryption. To take an example, in the `SSL` protocol, the compression is devoted to a higher level protocol which slows down the process.

Next section presents the algorithm of our scheme.

3.3 Implementation

```
Index = 256; Length = 9; Word = null; Limit = 12;
Initialise 256 inputs in the dictionary (eventually after a permutation)
//(a+b stands for concatenation)
REPEAT
        Read a symbol from the stream (read S)
        IF Word+S is already in the dictionary THEN
                Word = Word+S; Emit = false
        ELSE
                Output the (index of Word) XOR (PRS)  // Vernam cipher
                Index of (Word+S) = Index; Index++
                IF Length =  Limit THEN
                   Re-initialise the dictionary (Index=256, Length=9)
                ENDIF
                IF Index = 2*Length THEN Length++ ENDIF
                Word = S; Emit = true
        ENDIF
UNTIL no data found
IF Emit =  false THEN Output the (index of Word) XOR (PRS) // Vernam cipher
ENDIF
```

The implementation was just made as a proof-of-concept in C++ and using the LEDA library[†] which provides a sizable collection of data types and algorithms in a form which allows them to be used by non-experts. We have denoted by PRS a *pseudo-random (Boolean) sequence*.

3.4 Analysis

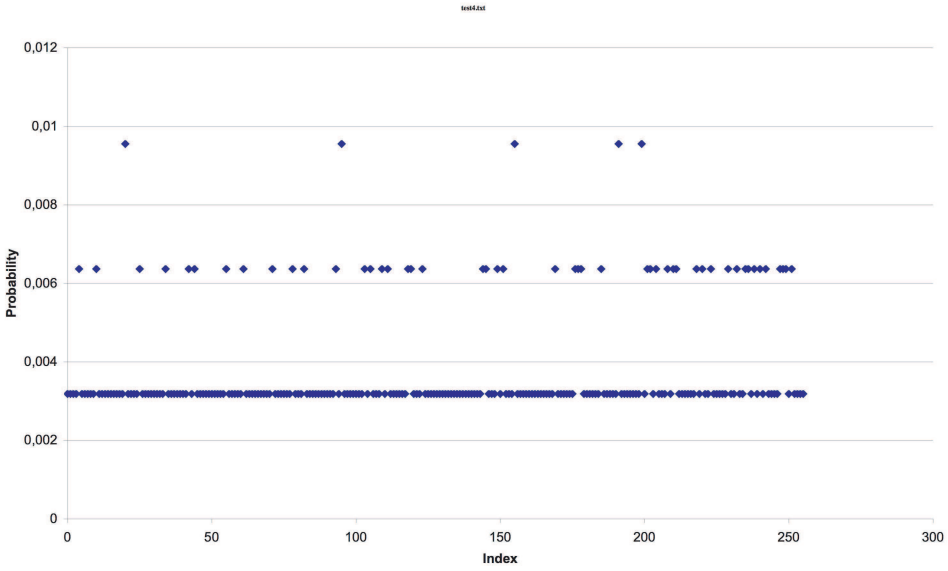

Fig. 1: Probabilities of the output of the scheme,

Though a plot of the output (see Figure 1) is rather encouraging, we were deceived while testing outputs with a χ^2 test for which the results were a little bit too weak. This may be explained by the rather bad choice of a "toy" linear feedback shift register for generating the PRS (just a polynomial of degree 7). We expect the result to be improved with the use of better pseudo-random generators like those we present in Section 4.

With the latter, further testing should be made according to the FIPS 140-2 standard [Tec97] which requires a pseudo-random generator to pass a number of statistical tests or the Marsaglia tests, a set of 23 very strong tests of randomness implemented in the Diehard program[‡].

4 Changing the pseudo-random sequence

In this section, we will focus on the pseudo-random sequences generated by cellular automata. We first recall the definition of binary cellular automata and their advantages for pseudo-random generation. We then explore the set of all the cellular automata rules to find out the most suitable for our purpose. As we will see in Section 4.2, pseudo-random sequences generated by binary cellular automata are not so good. So, we propose two different tracks to improve the quality of the sequences.

First of all, why do we consider cellular automata? One of their great interest is their intrinsic parallelism which permits to handle high rate pseudo-random generation. Cellular automata are also suitable for a programmable hardware implementation. Their target hardware model is the Field Programmable Gate Arrays (popularly known as FPGAs). FPGAs are now a popular implementation style for digital logic

[†] Avaible from `http://www.mpi-sb.mpg.de/LEDA`.

[‡] Avaible from `http://diehard.darwinports.com`.

systems and subsystems. These devices consist of an array of uncommitted logic gates whose function and interconnection is determined by downloading information to the device. When the programming configuration is held in static RAM, the logic function implemented by those FPGAs can be dynamically reconfigured in fractions of a second by rewriting the SRAM configuration memory contents [PB98]. Thus, the use of FPGAs can speed up the computation done by the cellular automata. Putting both remarks all together allows high-rate pseudo-random generation.

Secondly, the relationships between cellular automata and secret key cryptography is not new: in the context of symmetric key systems, cellular automata were first studied by [Wol85]. Unfortunately, as we will see in Section 4.2, elementary cellular automata are not suitable for cryptographic purpose. Different approaches were proposed later by Habutsu et al. [HNSM91], Nandi et al. [NKC94] and Gutowitz [Gut93]. With a slight modification of the model of cellular automata, one can get more promising results; it is the approach of Sipper et al. [ST96] and, more recently, of Seredynski et al. [SBZ04].

4.1 Cellular automata

In this section, we briefly recall the definition of cellular automata. We focus on the so called elementary cellular automata rules (also called Wolfram rules).

A *cellular automaton* (CA for short) is generally a bi-infinite array of identical cells which evolve synchronously and in parallel according to a local transition function. The cells communicate with their nearest neighbors. Here, we consider a ring of N cells which are indexed by $\mathbb{Z}_N$. All the cells are identical finite state machines with a finite number of states and a transition function which gives the new state of a cell according to its current state and the current states of its nearest neighbors.

Definition 2 *A* cellular automaton *is a finite array of identical cells indexed by $\mathbb{Z}_N$. Each cell is a finite state machine $C = (Q, f)$ where Q is a finite set of states and f a mapping $f : Q \times Q \times Q \to Q$.*

The mapping f, called *local transition function*, has the following meaning: the state of cell i at time $t+1$ (denoted by x_i^{t+1}) depends upon the state of cells $i-1$, i and $i+1$ at time t (the *neighborhood* of cell i of radius 1). Figure 2 illustrates one transition of a cellular automaton on a ring with 8 cells. We have the following equality which rules the dynamics of the cellular automaton:

$$x_i^{t+1} = f(x_{i-1}^t, x_i^t, x_{i+1}^t)$$

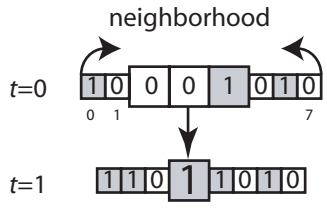

Fig. 2: Transition of cell 3 with rule 30 depicted.

For a fixed t, the sequence of all the values x_i^t for $i \in \mathbb{Z}_N$, is the *configuration* at time t. It is a mapping $c : \mathbb{Z}_N \to Q$ which assigns a state of Q to each cell of the cellular automaton. The sequence of configurations as pictured by Figure 3 is called a *time-space* diagram. Figure 3 depicts the evolution of a ring with $N = 8$ cells. On the top of Figure 3, we have depicted rule 30 with each transition illustrated by three adjacent squares representing the different preimages of f and on the bottom, their image by f. A 0 is painted white while a 1 is painted grey. On the bottom of Figure 3, we see the time-space diagram of the cellular automaton from the *initial configuration* at time $t = 0$ to time $t = 7$.

We will restrict ourselves to the case where $Q = \mathbb{F}_2$ and f is a Boolean predicate with 3 variables, also called an *elementary rule*. These cellular automata have been widely considered by Wolfram in [Wol86b]:

he considers the 256 different binary cellular automata and associates a natural number to each rule as follows:

$(x_{i-1}^t x_i^t x_{i+1}^t)$	111	110	101	100	011	010	001	000
x_i^{t+1}	0	0	0	1	1	1	1	0

The top line gives all possible preimages for f while the bottom line gives the images by f of the three binary values. Thus, f is fully specified by the 8-bit number written on the bottom line (00011110 in our example) which can be translated in basis 10 and then called the *rule* of the cellular automaton (as rule number 30 here). Equivalently, this rule can be considered as a Boolean function with (at most) 3 variables. Taking rule 30 again, its corresponding Boolean function is:

$$x_i^{t+1} = x_{i-1}^t \oplus (x_i^t \vee x_{i+1}^t)$$

with $\oplus$ denoting the Boolean XOR function and $\vee$ the classical Boolean OR function. Its equivalent formulation in arithmetic modulo 2 is:

$$x_i^{t+1} = \left(x_{i-1}^t + x_i^t + x_{i+1}^t + x_i^t x_{i+1}^t\right) \bmod 2$$

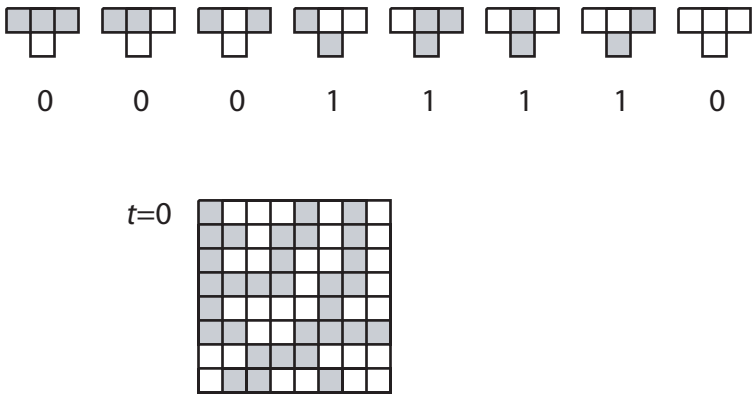

Fig. 3: Evolution of CA30 on a ring with $N = 8$ cells.

4.1.1 Equivalent rules

Since we will be dealing with pseudo-random generators, some rules are equivalent from this point of view by three transformations. To define the transformations, just recall that a transition is $f(x) = y$ with $x \in \mathbb{F}_2^3$ and $y \in \mathbb{F}_2$.

In the sequel, we denote by $\tilde{w}$ the mirror image of word $w = w_1 \dots w_n$, $\tilde{w} = w_n \dots w_1$ and by $\overline{w}$ the word obtained from w by exchanging the 0's by 1's (and conversely) $\overline{w} = \overline{w_1} \dots \overline{w_n}$.

The first transformation is called the *conjunctive* transformation which takes as an input rule r written in binary and returns $\bar{\tilde{r}}$. For instance, the conjunctive transformation turns rule 30 into rule 135.

The second transformation, called *reflexive* just changes the order of the x's by: $y_i = f(\tilde{x_i})$ for $i \in [\![0,7]\!]$. As an example, with reflexive transformation, rule 30 is changed into rule 86.

Last transformation combines both and is called *conjunctive-reflexive*: $y_i = \overline{f(\tilde{x_i})}$ for $i \in [\![0,7]\!]$ and it changes rule 30 into rule 149.

All these transformations keep the spectral values of the cellular automata dynamics and are thus statistically equivalent.

Next section describes how Wolfram proposed to generate pseudo-random sequences with cellular automata.

4.1.2 Pseudo-random generation with CA

More specifically in [Wol85, Wol86a], Wolfram uses a one-dimensional cellular automaton for pseudo-random bit generation by selecting the values taken by a single cell when iterating the computation of rule 30 from an initial finite configuration where the cells are arranged on a ring of N cells (see Figure 4). Mathematically, Wolfram claims the sequence $\{x_i^t\}_{t\geq 0}$ is pseudo-random for a given i. Wolfram studied this particular rule extensively, demonstrating its suitability as a high performance randomizer which can be efficiently implemented in parallel; indeed, this is one of the pseudo-random generators which was shipped with the connection machine CM2. Wolfram used classical randomness testing procedure from [Knu69]. In each case, the basic method consists in comparing an observed distribution with that calculated for a purely probabilistic sequence. He has been considering the block frequency distribution test for deciding whether all of the 2^n possible n-blocks should occur equally; the gap length distribution to compare with a binomial distribution. Wolfram also considered the distinct blocks distribution, the block accumulation distribution, the permutation frequency distribution, the monotone sequence length distribution and the maxima distribution. The pseudo-random sequence generated by rule 30 passed all of these tests.

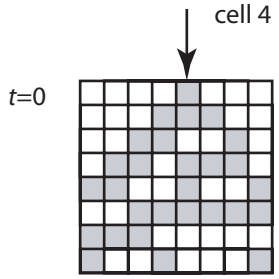

Fig. 4: Values taken by cell 4: 11011100.

Next section introduces the Walsh transform which is useful for studying the quality of the pseudo-random sequence generated.

4.1.3 Walsh transform

Walsh transform allows to compute the correlation between the inputs and the outputs of the iterations of a cellular automaton. One of the great interest of the Walsh transform is that its computation is even faster than computing a FFT (see [ER82] for instance).

Let us denote by $f(\underline{x})$ the value of function f at $\underline{x} = (x_0, x_1, x_2, \ldots, x_{n-1}) \in \mathbb{F}_2^n$ or, equivalently, $f(x)$, the value of f at $x = \sum_{i=0}^{n-1} x_i.2^i$, the decimal value corresponding to $\underline{x}$. Analogously, let $\underline{\omega} = (\omega_0, \ldots, \omega_{n-1}) \in \mathbb{F}_2^n$ and ω its corresponding decimal value. The *Walsh transform* of f is defined by:

$$S_f(\omega) = \sum_{x=0}^{2^n-1} f(x) \times (-1)^{\underline{x}\cdot\underline{\omega}}$$

with $\underline{x}\cdot\underline{\omega} = \sum_{i=0}^{n-1} x_i.\omega_i$ denoting the Cartesian product of the two binary vectors.

From the spectral values of the Walsh transform, one can recover the function f with the *inverse Walsh transform*:

$$f(x) = 2^{-n} \sum_{\omega=0}^{2^n-1} S_f(\omega) \times (-1)^{\underline{x}\cdot\underline{\omega}}$$

The Walsh transform has some interesting statistical properties. For instance, the value of the transform at point 0 equals the mean value of the function: $S_f(0) = E[f(x)] = 2^{n-1}$ (with E denoting the expected value). This property permits to test whether f is *balanced* (the number of 0's equals the number of 1's in the image domain of f).

In addition, Walsh transform is the main tool to study the *correlation-immunity* of a function.

t	1		2		3		4		5		conj	refl	c.r.
rule	cfg	val	cfg	val	cfg	val	cfg	val	cfg	val			
30	4	2	16	4	64	16	256	40	1024	80	135	86	149
60	0	0	0	0	0	0	0	0	0	0	195	102	153
86	1	2	1	4	1	16	1	40	1	80	149	30	135
90	0	0	0	0	0	0	0	0	0	0	165	90	165
102	0	0	0	0	0	0	0	0	0	0	153	60	195
105	0	0	0	0	0	0	0	0	0	0	105	105	105
135	4	2	16	4	64	16	256	40	1024	80	30	149	86
149	1	2	1	4	1	16	1	40	1	80	86	135	30
150	0	0	0	0	0	0	0	0	0	0	150	150	150
153	0	0	0	0	0	0	0	0	0	0	102	195	60
165	0	0	0	0	0	0	0	0	0	0	90	165	90
195	0	0	0	0	0	0	0	0	0	0	60	153	102

Tab. 1: "Good" rules after selection and their equivalent rules.

Definition 3 *A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is* k-th order correlation-immune *if, given n independant binary random variables $X_0, X_1, \cdots, X_{n-1}$ such that $Pr[X_i = 0] = Pr[X_i = 1] = \frac{1}{2}$ for $i \in [\![0, n-1]\!]$, then the random variable $Z = f(X_0, X_1, \cdots, X_{n-1})$ is independent from any random vector $(X_{i_1}, X_{i_2}, \cdots, X_{i_k})$, $0 \leq i_1 < \cdots < i_k < n$.*

Xiao and Massey [XM88] have characterized (k-th order) correlation-immunity with the Walsh transform. We recall this result in Theorem 3 in which the *Hamming weight* just counts the number of non-zero values in a vector.

Theorem 3 ([XM88]) *A function $f : \mathbb{F}_2^n \to \mathbb{F}_2$ is k-th order correlation-immune if and only if $S_f(\omega) = 0$ for all $\omega = (\omega_0, \omega_2, \cdots, \omega_{n-1}) \neq 0$ whose Hamming weight is at most k.*

For readers interested in a proof of Theorem 3, one can refer to [Zém00].

Actually, the idea of using Walsh transform to test pseudo-random generators comes from [Yue77]. In this paper, Yuen observed that a truly random sequence has an asymptotically flat Walsh power spectrum. This observation was used to devise a new test for randomness of the output of pseudo-random generators. It was an improvement of the tests described in Knuth [Knu69] who do not deal with *cryptographic* pseudo-random generation.

We have implemented this test for studying the set of all the 256 elementary cellular automata.

4.2 No correlation-immune elementary CA rule

We have used in [Mar06] a Walsh transform to study all the 256 elementary CA rules in order to find the best (non-linear) rules for generating pseudo-random sequences.

After removing all non-balanced rules by computing their Walsh transfom, we select rules f for which $S_f(0) = 4$; there are only 70 remaining balanced rules. Among those 70 rules f_i, we compute the maximum absolute value of the Walsh transform of the t^{th}-iterate of f_i at all the points $\underline{\omega}$ of Hamming weight 1 and we select the rules with a minimum spectral value. That is, we select rules f_i such that:

$$\min_{f_i} \max_{\omega=2^\ell} |S_{f_i^{(t)}}(\omega)|$$

where t denotes the iteration number and ω is the binary expansion of 2^ℓ for $\ell \in [\![0, 2t+1]\!]$, all the t bit vectors of Hamming weight one. Equivalently, we just test if, among the 70 balanced rules, there are some which are first order correlation-immune. After this, there are only the 12 remaining rules listed in Table 1 still containing linear rules which we recall in Table 2 in which the binary value encoding Wolfram's rule is written with the most significant bit on the rightmost part.

Boolean function	binary	decimal
$x_i \oplus x_{i+1}$	00111100	60
$x_{i-1} \oplus x_i$	01100110	102
$x_{i-1} \oplus x_{i+1}$	01011010	90
$x_{i-1} \oplus x_i \oplus x_{i+1}$	01101001	150
$\overline{x_{i-1} \oplus x_i}$	10011001	153
$\overline{x_i \oplus x_{i+1}}$	11000011	195
$\overline{x_{i-1} \oplus x_{i+1}}$	10100101	165
$\overline{x_{i-1} \oplus x_i \oplus x_{i+1}}$	10010110	105

Tab. 2: Linear rules and their corresponding Boolean functions.

Finally, if we remove the 8 linear rules, the best remaining rules are 30, 135, 86 and 149 which are all together equivalent by conjunctive, reflexive and conjunctive-reflexive transformations. None of them is first order correlation-immune nor correlation-immune as well. Thus, we state that:

Theorem 4 *There is no non-linear correlation-immune elementary cellular automaton.*

And, according to Theorem 2, we can state that:

Corollary 1 *There is no elementary cellular automaton which can serve as pseudo-random sequence generator.*

That is the reason why pseudo-random sequences generated in this way can be reversed by a correlation attack like the one proposed in [MS91], although pseudo-random sequences generated by rule 30 (or, equivalently by rules 86, 135 and 149) passed classical statistical tests like those proposed in [Knu69]. The attack proposed by Meier and Staffelbach in [MS91] is simple. It suffices to write the way to obtain the sequence generated by rule 30:

$$x_i^{t+1} = x_{i-1}^t \oplus (x_i^t \vee x_{i+1}^t) \tag{1}$$

and to use the partial linearity to rewrite equation 1 as:

$$x_{i-1}^t = x_i^{t+1} \oplus (x_i^t \vee x_{i+1}^t) \tag{2}$$

From equation (1), the cell values corresponding to the pseudo-random sequence are built from a triangle in the time-space diagram. And, from equation (2), one can guess the values of the leftmost part of the triangle (or equivalent partial configurations) and then find randomly an initial configuration for generating the pseudo-random sequence, exploiting the correlation.

Despite these weaknesses, we propose to improve the sequences generated by cellular automata to combine it with the LZ 78 compression process.

4.3 Improving the randomness

So, does Theorem 4 annihilate any hope to design a good pseudo-random generator by the means of cellular automata? Not necessarily. In this section, we will propose two different approaches. One is designed to counter Meier and Staffelbach attack and the second one changes the way pseudo-random sequences are generated.

4.3.1 Countering the attack

An idea to prevent the attack on the pseudo-random sequence proposed by Meier and Staffelbach is simply to change the way we extract the pseudo-random sequence from the sequence of configurations generated by the cellular automaton. Instead of taking the sequence $\{x_i^t\}_{t\geq 0}$ as proposed by Wolfram, we select a time-constructible function f and we consider the sequence $\{x_{f(i)}^t\}_{t\geq 0}$ for a suitable f.

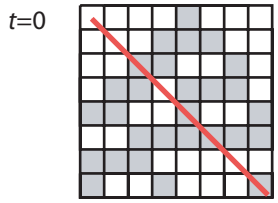

Fig. 5: Values taken by the cells according to f (represented with a thick line): 00101101.

Such functions, in the field of cellular automata are called *signals* and have been widely studied by Mazoyer and Terrier, see [MT99] for example. Just to fix the ideas, we have proposed in Figure 5 to take the cells values with a function f which moves to the right, one cell at a time. It is one of the simplest signal which can be defined on cellular automata.

Next section proposes a completely different approach. The idea is to use a set of rules for the cellular automaton dynamics instead of a single rule as in the original *uniform* model.

4.3.2 The cellular programming approach

Tomassini and Sipper [ST96] proposed to use non-uniform cellular automata for generating better pseudo-random sequences. In this model, each cell may contain a different rule (the cellular automaton becomes then *non-uniform*) and the rules are obtained by an evolutionary approach (by a genetic algorithm). They have designed a *cellular programming* algorithm for cellular automata to perform computations, and have applied it to the evolution of pseudo-random sequence generators. Their genetic algorithm uses Koza's *entropy* E_h. E_h measures the entropy for the set of k^h probabilities of the k^h possible subsequences of length h. It is defined by:

$$E_h = -\sum_{j=1}^{k^h} p_{h_j} \log_2 p_{h_j}$$

where k denotes the number of possible values per sequence position (in our case, the cellular automata states), h a subsequence length and p_{h_j} is a measured probability of occurrence of a sequence h_j in a pseudo-random sequence. The entropy achieves its maximal value $E_h = h$ when the probabilities of the k^h possible sequences of length h are all equal to $1/\ell^h$, where ℓ^h denotes a number of possible states of each sequence.

Tomassini and Sipper have selected four rules of radius 1 for use in non-uniform cellular automata. The best rules selected by the genetic algorithm were rules 90, 105, 150 and 165 (which are all linear, a drawback for certain attacks).

A series of tests (including χ^2 test, serial correlation coefficient, entropy and Monte Carlo, but no correlation-immunity analysis) were made with good results, showing that co-evolving generators are at least as good as the best available CA randomizer. The drawback here is that the authors also use elementary rules which we proved to be not correlation-immune. This was further investigated in [SBZ04].

Following the same kind of approach, Seredynski et al. in [SBZ04] have generalized the selection process to radius 2 rules. They use then both radius 1 and radius 2 rules in non-uniform cellular automata. The rules selected by their genetic algorithm were 30, 86, 101 and 869020563, 1047380370, 1436194405, 1436965290, 1705400746, 1815843780, 2084275140 and 2592765285.

Their new set of rules was tested by a number of statistical tests required by the FIPS 140-2 standard [Tec97] and the Marsaglia tests implemented in the Diehard program but no correlation-immunity analysis was made. And, for both approaches, it is recalled in [Zém00] that it can be dangerous to combine "bad rules".

5 Discussion

Although not truly pseudo-random (but this is also not a pseudo-random generator), the output of our compression and encryption scheme is encouraging if we look at the typical output depicted by Figure 1. Further study should be made with the help of a better pseudo-random generator based on cellular automata with classical tests and a fine tuning of all the parameters. In order to avoid correlated cellular automata, the best thing may be to combine the cellular programming approach and to consider the sequence of cells selected by a time-constructible function. The implementation has been started but is not yet completed.

The use of compression and encryption mixed together should increase the bandwidth, decrease the latency as well as it also might decrease the energy consumption required for the same purpose when using encryption after compression for mobile devices or RFID.

Last but not least, there is currently a great challenge in the use of a good pseudo-random generators; not only for use as a key for a Vernam-type cipher but also for computing a cryptographic hash function. Actually, from the collision attacks on MD5 and other hash function (see [Kli05] and [WY05]), one could use the improvements proposed by [DGV91] and later by [MZI98] of the hash function based on cellular automata which was originally suggested by Damgård in [Dam89]. And, in order to design a secure cryptographic hash function, we need to have a robust pseudo-random generator.

References

[BM84] M. Blum and S. Micali. How to generate cryptographically strong sequences of pseudo-random bits. *SIAM J. Comput*, 13:850–864, 1984.

[Cra98] R.E. Crandall. Comcryption. In *RSA Data Security Conference*, 1998.

[Dam89] I. B. Damgård. A design principle for hash functions. In *Crypto'89*, number 435 in Lecture Notes in Computer Science, pages 416–427. Springer Verlag, 1989.

[DGV91] J. Daemen, R. Govaerts, and J. Vandewalle. A framework for the design of one-way hash functions including cryptanalysis of Damgård's one-way function based on a cellular automaton. In *ASIACRYPT'91*, Lecture Notes in Computer Science. Springer Verlag, 1991.

[ER82] D.E. Elliott and K.R. Rao. *Fast transforms, algorithms, analysis, applications*. Academic press, 1982.

[Gut93] H. A. Gutowitz. Cryptography with dynamical systems. In E. Goles and N. Boccara, editors, *Cellular Automata and Cooperative Phenomena*. Kluwer Academic Press, 1993.

[HNSM91] T. Habutsu, Y. Nishio, I. Sasase, and S. Mori. A secret key cryptosystem by iterating a chaotic map. In *EUROCRYPT*, pages 127–140, 1991.

[Kli05] V. Klima. Finding MD5 collisions - a toy for a notebook, 2005.

[Knu69] D.E. Knuth. *Seminumerical Algorithms*. Addison Wesley, 1969.

[Kol65] A.N. Kolmogorov. Three approaches to the quantitative definition of information. *Problemy Pederachi Informatsii*, 1:3–11, 1965.

[LV93] M. Li and P. Vitányi. *An introduction to Kolmogorov complexity and its applications*. Texts and monographs in computer science. Springer Verlag, 1993.

[Mar04] B. Martin. *Codage, cryptologie et applications*. Presses Polytechniques et Universitaires Romandes, 2004.

[Mar06] B. Martin. A Walsh exploration of Wolfram CA rules. In *International Workshop on Cellular Automata*, pages 25–30, Hiroshima University, Japan, sep 2006.

[MS91] W. Meier and O. Staffelbach. Analysis of pseudo random sequences generated by cellular automata. In *EUROCRYPT '91*, Lecture Notes in Computer Science. Springer Verlag, 1991.

[MT99] J. Mazoyer and V. Terrier. Signals in one-dimensional cellular automata. *Theoretical Computer Science*, 217(1):53–80, 1999.

[MZI98] M. Mihaljevic, Y. Zheng, and H. Imai. A cellular automaton based fast one-way hash function suitable for hardware implementation. In *Public Key Cryptography*, volume 1431 of *Lecture Notes in Computer Science*, pages 217–234. Springer Verlag, 1998.

[NKC94] S. Nandi, B. K. Kar, and P. P. Chaudhuri. Theory and applications of cellular automata in cryptography. *IEEE Trans. Computers*, 43(12):1346–1357, 1994.

[PB98] R. B. Porter and N. W. Bergmann. Evolving FPGA based cellular automata. In B. McKay, X. Yao, C. S. Newton, J-H. Kim, and T. Furuhashi, editors, *SEAL*, volume 1585 of *Lecture Notes in Computer Science*, pages 114–121. Springer, 1998.

[Res01] E. Rescorla. *SSL and TLS: Designing and Building Secure Systems*. Addison-Wesley, Reading MA, 2001.

[Sal98] D. Salomon. *Data compression, the complete reference*. Springer Verlag, 1998.

[SBZ04] F. Seredynski, P. Bouvry, and A. Y. Zomaya. Cellular automata computations and secret key cryptography. *Parallel Comput.*, 30(5-6):753–766, 2004.

[ST96] M. Sipper and M. Tomassini. Co-evolving parallel random number generators. In *Parallel Problem Solving from Nature – PPSN IV*, pages 950–959, Berlin, 1996. Springer Verlag.

[Sta06] W. Stallings. *Cryptography and Network Security*. Prentice-Hall, 4th. edition, 2006.

[Tec97] National Institute Of Standards Technology. FIPS publication 140-2, Security requirements for cryptographic modules. US Gov. Printing Office, Washington, 1997.

[Wol85] S. Wolfram. Cryptography with cellular automata. In *CRYPTO 85*, Lecture Notes in Computer Science. Springer Verlag, 1985.

[Wol86a] S. Wolfram. Random sequence generation by cellular automata. *Advances in applied mathematics*, 7:123–169, 1986.

[Wol86b] S. Wolfram. *Theory and applications of cellular automata*. World Scientific, Singapore, 1986.

[WY05] X. Wang and H. Yu. How to break MD5 and other hash functions. In *EUROCRYPT*, pages 19–35, 2005.

[XM88] G-Z. Xiao and J. L. Massey. A spectral characterization of correlation-immune combining functions. *IEEE Trans. on Information Theory*, 34(3):569–, 1988.

[Yao82] A.C. Yao. Computational information theory. In *Proceedings of the IEEE*, 1982.

[Yue77] C-K. Yuen. Testing random number generators by Walsh transform. *IEEE Trans. Computers*, 26(4):329–333, 1977.

[Zém00] G. Zémor. *Cours de cryptographie*. Cassini, 2000.